Simulation Technology & Applied Research, Inc.

11520 N. Port Washington Rd., Suite 201 Mequon, WI 53092-3432
P: 1.262.240.0291 F: 1.262.240.0294 W: www.staarinc.com

Computer-Aided Design and Optimization of High-Performance Vacuum Electronic Devices

SBIR Phase 2
Progress Report 1

(covering period 11/1/2005-1/31/2006)

Contract/Purchase Order Number: N00014-05-C-0375
Simulation Technology & Applied Research Report Number: 06-SBIR-ONR-T1

Prepared for the Office of Naval Research

February 21, 2006

John F. DeFord, Ben Held, and Liya Chernyakova
Simulation Technology & Applied Research, Inc.
11520 N. Port Washington Rd., Suite 201
Mequon, WI 53092

P: 1-262-240-0291 x102 F: 1-262-240-0294 E: john.deford@staarinc.com

John Petillo
Scientific Applications International Corporation
Suite 130, 20 Burlington Mall Rd.
Burlington, MA 01813

P: 1-781-221-7615 F: 1-781-270-0063 E: jpetillo@bos.saic.com

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
21-02-2006	Interim	Oct., 2005 - Jan., 2006

4. TITLE AND SUBTITLE
Computer-Aided Design and Optimization of High-Performance Vacuum Electronic Devices

5a. CONTRACT NUMBER
N00014-05-C-0375

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
J. F. DeFord, B. Held, L, Chernykova
Simulation Technology & Applied Research, Inc.

J. Petillo
Scientific Applications International Corporation

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Simulation Technology & Applied Research, Inc.
11520 N. Port Washington Rd., Suite 201, Mequon, WI 53092

8. PERFORMING ORGANIZATION REPORT NUMBER
06-SBIR-ONR-T1

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Office of Naval Research
875 North Randolph St. Suite 1425
Arlington, VA 22203-1995

10. SPONSOR/MONITOR'S ACRONYM(S)
ONR

11. SPONSORING/MONITORING AGENCY REPORT NUMBER
001AC

12. DISTRIBUTION AVAILABILITY STATEMENT
Unlimited/Unclassified

13. SUPPLEMENTARY NOTES

14. ABSTRACT
Project activities during the previous three months include the acquisition and setup of a LINUX cluster for use in optimization studies and the design and initial implementation of the optimization library, including support for the differential evolution and multi-directional search algorithms. Initial testing of the algorithms on several analytic functions has validated the implementations. Work on the implementation of user-interface support for the optimization library in the Analyst product has also begun, with the initial implementations of all panels now complete.

15. SUBJECT TERMS
optimization,LINUX cluster,differential evolution,multi-directional search

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			John F. DeFord
UU	UU	UU	UU	9	19b. TELEPONE NUMBER (*Include area code*) (262) 240-0291 x102

Abstract

Project activities during the previous three months include the acquisition and setup of a LINUX cluster for use in optimization studies and the design and initial implementation of the optimization library, including support for the differential evolution and multi-directional search algorithms. Initial testing of the algorithms on several analytic functions has validated the implementations. Work on the implementation of user-interface support for the optimization library in the Analyst product has also begun, with the initial implementations of all panels now complete.

Executive Summary

The following items have been accomplished in the first three months of the project:

- LINUX cluster has been procured and is now operational at STAR.
- Basic implementations of multi-directional search (MDS[1]) and differential evolution (DE[2]) algorithms is complete.
- Initial testing of MDS and DE implementations is complete.
- Work on the Analyst user interface to the optimization library has begun.

LINUX Cluster

As part of the project startup we built a cluster that will be used for design optimization. The current cluster consists of 4 dual processor computers. Each computer utilizes the Tyan S2892G3NR mainboard, dual AMD Opteron 248 (2.2GHz, 1MB Cache) processors, 8 GB RAM (4 each - DDR400 2GB ECC/Registered – can be expanded to 16 GB each), dual Seagate 250GB Serial ATA 7200rpm hard drives, DVD-RW drive, all enclosed in a Cooler Master STC-T01-UWK Stacker case. The OS is the latest version of Redhat Fedora Core 4 (64-bit version). The computers are connected using a single gigabit Ethernet (Cat 5e) using a dedicated 16-port gigabit switch. The LAM MPI implementation (version 7.1.1-3) is being utilized for inter-process communication.

On tests of a parallel LU solver written by STAR, the cluster has achieved speeds in excess of 4000 MFLOP when all 8 processors are employed, and we expect it to dramatically reduce collector optimization times when used with MICHELLE (MICHELLE is a serial code, but we will utilize the cluster to run up to eight simultaneous jobs during an optimization).

Implementation of Multi-Directional Search and Differential Evolution Algorithms

The initial implementations of both DE and MDS are now complete, and integration into Analyst is underway. The implementations are composed of a set of C++ classes that will eventually be combined into a library that can be delivered to NRL, and also utilized in both the Voyager GUI for MICHELLE (where it will interface to the Java system), and the Analyst package.

The optimization package is architected so that it may be used to control a generic optimization process. As such, method-specific functions are declared as virtual or pure virtual in base classes, and are overloaded as necessary in derived classes that are specialized to a particular method. This architecture will make it very easy to add other methods to the library, without having to modify core functionality like process control, previous result lookup, and interactions with the analysis package. The classes are briefly described in Appendix 1.

---

[1] V. Torczon, *Multi-Directional Search: A Direct Search Algorithm for Parallel Machines*, Ph. D. thesis, Rice University, Houston, TX, May, 1989.

[2] R. Storn and K. Price, "Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces," *J. Global Optim.*, **11**, 1997, pp. 341-359.

MDS Algorithm

This approach is a direct search simplex method that is closely related to the Nelder-Mead method[3] in which a non-degenerate simplex of dimension n+1 is updated (for an n-dimensional parameter vector) at each step. The volume enclosed by the simplex reduces until it encloses an extremum of the objective function.

A step in the process begins with stored values of the simplex vertices (parameter vectors) and the associated objective function values. The vertex with the best (minimum) value of the objective function is identified, and a set of n search directions is defined by the edges that connect the best vertex to the remaining vertices in the simplex. The length of each edge defines the length of the associated step in that direction, i.e., the new sample points are obtained by "reflecting" each vertex about the best vertex, with the connecting edge defining the reflection plane normal. The new sample points, together with the previous best vertex, form a new simplex that is accepted for the next iteration if at least one of its vertices has a better objective function value than does the previous best vertex. If the simplex is not accepted there are is a simple set of expansion and/or contraction steps (wherein the search directions are maintained but the step-size is changed) that are performed to find an acceptable new simplex. The process terminates when the simplex nodes, and the corresponding goal function values, become "close enough". More precisely, the two criteria are:

$$\max\left(F\left(\vec{X}_{best}\right)-F\left(\vec{X}_{i}\right)\right)|_{i=1\ldots NES}\leq\delta$$

$$\sqrt{\frac{\sum_{k=1}^{NES}\|\vec{X}_{k}-\vec{X}_{mean}\|^{2}}{NES}}\leq\delta$$

where δ is a user defined parameter, and NES is the number of experiments in one iteration.

One advantage this method has over Nelder-Mead and some other direct search approaches is that it is inherently parallel, because the processing and function evaluations associated with the different search directions are independent and can be performed simultaneously on different processors. This feature will be exploited later on in the project when we enable Analyst to distribute independent MICHELLE analyses on individual processors of a cluster.

DE Algorithm

This method comes from a class of algorithms based upon evolutionary principles. To start the process, an initial "population" of random vectors $\{\mathbf{p}_{k,0}\}$ is created that all satisfy the parameter constraints. At each iteration (called a "generation") of the process, new vectors are obtained from the previous set using the following concepts:

- *Mutation*. A new vector is formed via a combination of existing vectors of the form

$$\mathbf{v}_{i,G+1}=\mathbf{p}_{i,G}+\alpha\left(\mathbf{p}_{k,G}-\mathbf{p}_{l,G}\right)$$

[3] Nelder, J. A. and Mead, R. "A Simplex Method for Function Minimization." *Comput. J.* **7**, 308-313, 1965.

- *Recombination (also called crossover)*. A candidate "child" vector is formed by taking some (randomly selected) parameter values directly from the parent $\mathbf{p}_G$, and the rest from the differential combination vector $\mathbf{v}_G$, i.e.,

$$\mathbf{u}_{i,G+1} = \begin{cases} \mathbf{v}_{i,G} \ , i \in S \\ \mathbf{p}_{i,G} \ , i \notin S \end{cases}$$

- *Selection*. A parent vector is replaced with a child vector if the objective function is reduced. Otherwise, additional children are created and tested until either one is found that reduces the objective function or some maximum number of offspring is reached. If no child is more "fit" than the parent, the parent passes to the new generation (if they are not eliminated by the aging criterion below).
- *Aging*. A vector can only "survive" for a limited number of generations, regardless of its "fitness".

In addition to the basic DE algorithm, we implemented several variations of how the subsequent generation is constructed[4].

Variation Name	Expression
A	$Next[n] = Cur[n] + F * (Cur[rand_1] - Cur[rand_2])$
B	$Next[n] = Best + F * (Cur[rand_1] - Cur[rand_2])$
C	$Next[n] = Cur[rand_3] + F * (Cur[rand_1] - Cur[rand_2])$
D	$Next[n] = Cur[n] + F * (Best - Cur[n]) + F * (Cur[rand_1] - Cur[rand_2])$

$Next[n]$ – next n-th trial vector.
$Cur[n]$ – current n-th trial vector.
$Best$ - best vector in the trial set.
$rand_i$ - randomly selected vector indexes in the previous random set, $n \neq rand_i$
F – user defined coefficient $0 \leq F \leq 1$.

Goal Functions

Goal functions and constraints (with the exception of simple range constraints) will be defined in terms of Python functions. The user interface will present the user with a set of pre-defined goal functions, and also provide the ability to create/import/edit goal function and constraint scripts.

Algorithm Testing

We have tested the algorithms on a variety of analytic functions, including:

1. Simple quadratic in two dimensions.
2. A discrete function of two parameters of the form:

$$R[x,y] = \begin{cases} rand(0,1)_{i,j} & -100 \leq (i = \text{int}(x/dx)) \leq 100, -100 \leq (j = \text{int}(y/dy)) \leq 100 \\ \infty & otherwise \end{cases}$$

[4] R. Storn and K. Price, differential evolution "c" code, http://www.icsi.berkeley.edu/~storn/de36.c.

where $rand(0,1)_{i,j}$ is a random number in the range [0,1] that is generated for each *i,j* position on the grid.

3. The square root of *R* defined above.
4. The square of R defined above.
5. A sum of squares of thirty variables.

These functions were chosen to stress the algorithm and reveal any problems in the implementation, not because they are representative of real-world design optimization problems. Based on these tests we believe that all of the variants of DE are now working properly, and testing of MDS is underway. Testing on design problems will begin as soon as the implementation in Analyst is complete, and this is expected within the next few weeks.

User Interface Development

The user interface in Analyst to the optimization library will take the form of a "wizard", which is a set of panels that are navigated via "Next" and "Back" buttons. This format allows control over the setup process, thereby simplifying it for the user. The panels will walk the user through the following steps:

1. *Parameter selection*. The user is presented with a list of all parameters of the model (these will include geometric, material, and solver setup parameters). Parameters are included/excluded via a checkbox next to the name, and the user can also specify a short codename for each parameter that is used in the constraint and goal function definitions.
2. *Constraint definition*. This includes defining both a range of validity for each selected parameter, and also an optional Python function that further restricts the domain.
3. *Goal function definition*. A Python function that takes a result database and list of geometric parameters and returns a real number. Several predefined functions will be defined, and the user will also be able to define their own.
4. *Setting of optimization control parameters*. Selection of the algorithm, and definition of values that control the algorithm, e.g., the maximum number of analyses.
5. *Execution of optimization*. Progress information will be displayed in tabular and graphical formats, and various abort options will be available to allow termination of the process if desired.

The initial implementations of some of these panels is shown in Figs. 1-3.

Next Phase

Work in the next 3 months will include:

- Completion of user interface to optimization capabilities in Analyst.
- Continued testing and refinement of MDS and DE algorithms.
- Initial optimizations of a collector with MICHELLE.

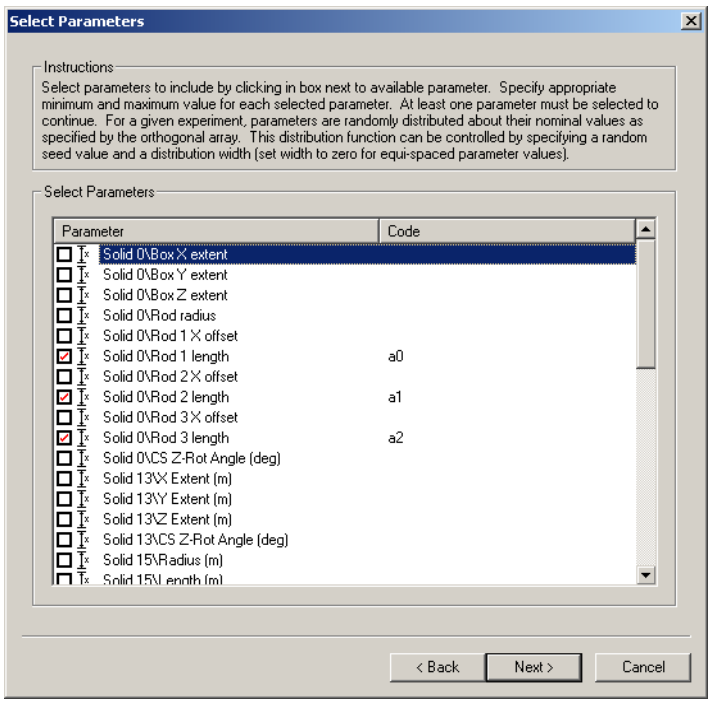

Fig. 1. Parameter selection panel.

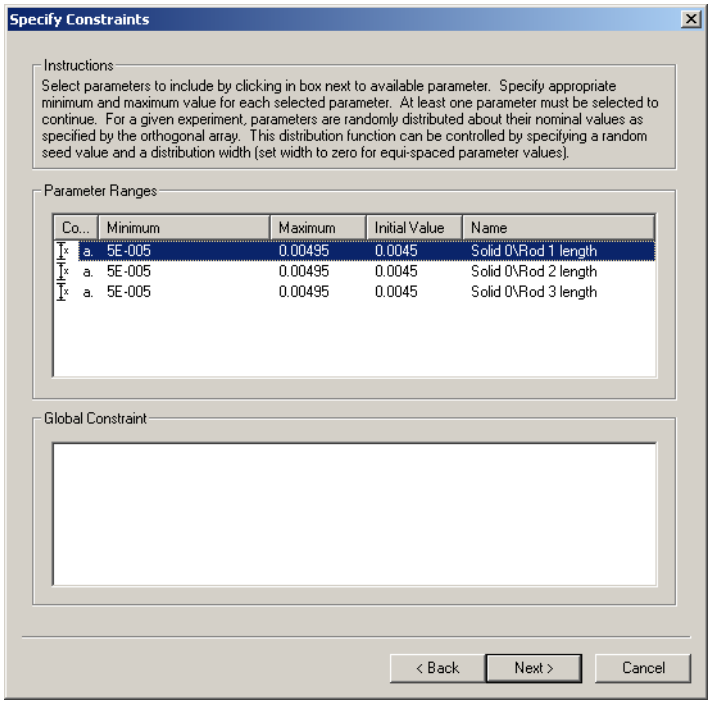

Fig. 2. Parameter constraint definition panel.

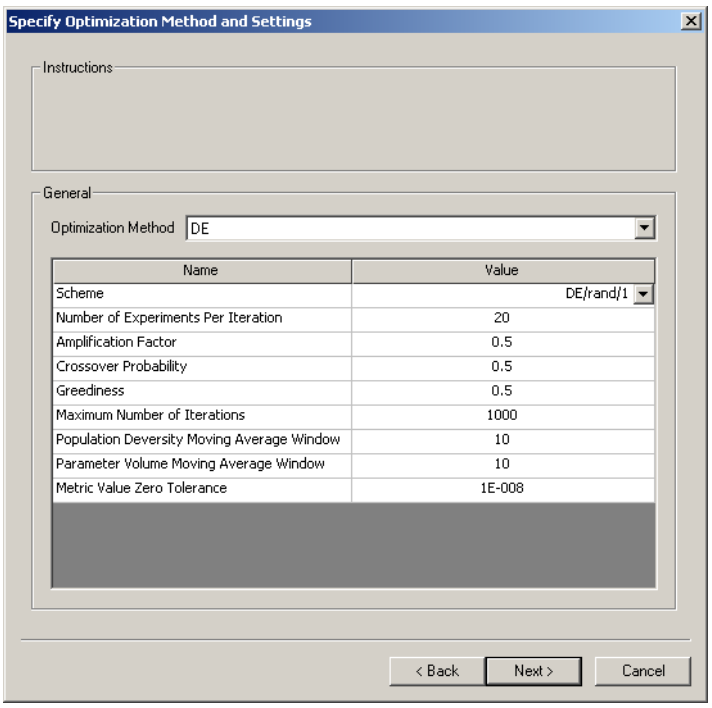

Fig. 3. Algorithm control panel.

Appendix 1: Software Description

The optimization algorithms were implemented in the C++ programming language, and will later be made into a library that can be called from either Analyst or the Voyager GUI. In order to implement optimization in the most general terms a class hierarchy was created that will support not only MDS and DE, but other methods as well. The current implementation has support for arbitrary numbers of parameters, range limits on parameters, and the ability to define rectangular volumes in parameter space that are "out-of-bounds". The important classes in this hierarchy are listed in Table 1.

Table 1. Optimization classes.

Class Name	Purpose
SOptOptimizer	Responsible for procedures common for all optimization algorithms. Implements optimization, submitting trial set for analysis and retrieving results, submitting information to the program log, storing necessary information on completed trials in the program database.
SOptDEOptimizer	Implements DE optimization algorithms. Inherits from SOptOptimizer, overloads functions responsible for initializing first trial set, analyzing current trial and creating next trial set, check for stopping criteria.
SOptMDSOptimizer	Implements MDS optimization algorithms. Inherits from SOptOptimizer, overloads functions responsible for initializing first trial

	set, analyzing current trial and creating next trial set, check for stopping criteria.
SOptExperimentMgr	Interface for the program that runs trial vector analysis. Interfaces functions for evaluating set of goal functions from set of trial vectors, function that initializes trial vector variables, implements functions responsible for creating single trial vector and single trial variable.
SOptLog	Interface to store general information on optimization, interfaces Log, Message, Warning, Error, Fatal functions.
SOptOutputMgr	Interface for storing trial sets in some predefined database. Interface functions responsible for adding new table definition, adding data to and retrieving from existing table.
SOptExperiment	Stores data related to one trial vector. Implements functions for creating, arithmetical manipulations, and validity check as well as direct setting/retrieving variable/goal function values and trial status.
SOptControlParams	Stores/retrieves user defined optimization control parameter.
SOptExperimentVariableValues	Responsible for storing/retrieving/arithmetical operations and validity check over trial vector component values.
SOptExperimentVar	Responsible for storing/retrieving/arithmetical operations and validity check over one component of trial vector.
SOptVariableConstraint	Implements single constraint requirement for one trial vector component to meet.
SOptExperimentCollection	Implements internal storage for trial vector sets.